Prototypage Matériel-Logiciel de Systèmes Intégrés avec l'architecture RISC-V

N. Ait Said, M. Benabdenbi, G. Villanova Magalhães

Univ. Grenoble Alpes, CNRS, Grenoble INP*, TIMA, 38000 Grenoble, France
* Institute of Engineering Univ. Grenoble Alpes

Contact : { noureddine.ait-said | mounir.benabdenbi | gabriel.villanova }@univ-grenoble-alpes.fr

Cet article présente un nouvel enseignement en cours de mise en place à Grenoble INP dont l'objectif est de former les étudiants à la conception conjointe matérielle-logicielle de systèmes intégrés. L'objectif principal de ce nouvel enseignement est de donner à l'étudiant une vision plus claire de l'interaction entre le matériel et le logiciel et de l'impact d'un changement de l'un sur l'autre. La plateforme utilisée est libre de droit (Rocket Chip développée à Berkeley), elle est basée sur l'utilisation d'un processeur récent, d'architecture RISC-V et elle permet de faire du prototypage rapide et/ou d'aller jusqu'à l'implémentation physique sur FPGA ou ASIC. Cette flexibilité en fait un bon candidat comme outil pédagogique.

I. Introduction

La conception des systèmes intégrés est basée sur la conception conjointe du matériel et du logiciel, appelée aussi co-conception ou co-design. Dans le cadre de la formation aux métiers de la conception en microélectronique, pour ce qui est du co-design de système on peut faire les constats suivants :

1. l'enseignement des matières liées à la conception des systèmes intégrés embarqués est souvent fait de telle manière que les deux volets logiciel (SW) et matériel (HW) sont traités séparément. L'étudiant, par conséquent, arrive difficilement à faire le lien et comprendre l'impact de l'un sur l'autre,
2. les plateformes utilisées pour la conception et le prototypage rapide, sont généralement soit propriétaires, soit obsolètes vis à vis de l'évolution des systèmes intégrés dans le monde,
3. l'utilisation de plateformes et outils de CAO propriétaires introduit des contraintes et des informations qui brouillent le message principal et limitent la compréhension de l'étudiant. Celui-ci en plus des notions principales doit intégrer l'usage des outils pour des technologies cibles.

A l'école d'ingénieurs PHELMA de Grenoble INP, nous utilisions jusqu'à présent la plateforme de conception et simulation SoCLib (1) pour enseigner la conception conjointe de systèmes intégrés multi processeurs (2). Cette plateforme est basée sur la simulation en SystemC d'un système constitué d'une interconnexion de composants modélisés en SystemC (processeurs MIPS, ARM, ..., mémoires, périphériques, ...). Les simulations sont précises au cycle près et bit près (Cycle Accurate, Bit Accurate ou CABA). Cette plateforme est bien adaptée pour l'enseignement néanmoins les modèles de simulation n'étant pas synthétisables, il faut se procurer auprès des vendeurs d'IP les modèles adéquats et les assembler pour aller vers l'implémentation sur FPGA ou ASIC. De plus, à notre connaissance, la bibliothèque SoCLib ne contient pas de modèles de processeurs récents.

Article publié par EDP Sciences et disponible sur le site https://www.j3ea.org ou https://doi.org/10.1051/j3ea/20191016

Pour pallier ces limitations et répondre aux points cités plus haut, on expérimente cette année un nouvel enseignement à destination d'étudiants de troisième année d'école d'ingénieur de la filière Systèmes Electroniques Intégrés possédant les prérequis de base en conception matérielle et logicielle. Cet enseignement est basé sur l'utilisation de l'architecture de processeurs libre RISC-V (3), et sur la plateforme de conception Rocket Chip (4), qui est à la fois *open source*, récente, bien maintenue, et qui suit l'évolution de la spécification standard. Le fait que RISC-V soit libre de droits, apporte un certain nombre d'avantages qui font que sa popularité devient de plus en plus croissante dans l'industrie (Nvidia, Western Digital, Microsemi ...) ainsi que dans le monde de la recherche scientifique. De plus Rocket Chip permet de générer et d'étudier des architectures hautement paramétrables. Cela permet aux étudiants d'approfondir leurs connaissances et de mettre en pratique les concepts théoriques acquis en cours et en travaux dirigés.
Dans cet article nous décrivons dans la section II, l'architecture, la plateforme de prototypage, et le flot de conception. La section III par la suite propose un bref survol des travaux pratiques utilisant cette plateforme.

II. Architecture et plateforme de prototypage utilisées

L'enseignement proposé repose sur l'architecture RISC-V (prononcé « RISC five »), dont le jeu d'instruction (instruction set architecture ou ISA) appartient à la famille des processeurs à jeu d'instructions réduit (RISC). RISC-V est une architecture ouverte et libre, ses spécifications sont ouvertes et elle peut être implémentée et utilisée librement pour l'enseignement, la recherche et l'industrie (3)-(5).

Bien que l'architecture de processeurs RISC-V soit *open-source*, il existe des implémentations (processeurs) commercialisées, comme par exemple, les processeurs Codix-Bk de chez Codasip. Il existe aussi des implémentations *open-source* telles que Rocket core (6), BOOM core (Berkley Out-of-Ordre Machine) (7), PULPino (8).

Une étude technique d'un ensemble d'outils a été réalisée et a conduit au choix de la plateforme Rocket Chip comme plateforme de développement SW/HW, et au choix du Rocket core comme processeur principal.

Flot de conception

La plateforme Rocket Chip est constituée entre autres d'un générateur de SoC, qui peut être utilisé pour générer différentes architectures matérielles. Le flot est représenté dans la figure 1. On trouve dans ce flot des :

1. Composants hardwares : décrits soit en CHISEL (9), un langage de description matérielle (Constructing Hardware in a Scala Embedded Language), soit en langage Verilog. L'ensemble peut être compilé et synthétisé pour implémentation soit sur FPGA/ASIC avec des outils industriels, soit traduit en un émulateur de hardware (C Emulator, ou verilated emulator) exécutable sur des machines PC (x86). Le résultat correspond à un système intégré matériel capable de faire fonctionner n'importe quel code logiciel RISC-V,
2. Composants softwares : compilés, assemblés et liés pour construire un binaire destiné à être exécuté sur les machines cibles RISC-V. Ces binaires peuvent aussi bien s'exécuter en bare-metal que s'appuyer sur un système d'exploitation temps réel comme FreeRTOS ou sur un OS normal tel que Linux.

Pour exécuter le binaire produit, le simulateur d'architecture de jeu d'instructions Spike (10) est utilisé. C'est un programme de simulation qui s'exécute sur une machine PC x86, qui vérifie que le binaire respecte bien la spécification officielle. Comme Spike ne tient pas

compte de certaines spécificités du matériel (initialisation de points mémoire, périphériques personnalisés …), l'émulateur matériel généré (C Emulator), est utilisé pour émuler les comportements du vrai hardware. Il s'agit d'un émulateur Cycle Accurate Bit Accurate (CABA), qui sert à mesurer les temps d'exécution des programmes en nombre de cycles, et qui permet de visualiser les signaux internes du/des processeurs et des périphériques personnalisés. Il permet ainsi aux étudiants de développer, tester et déboguer leurs propres périphériques écrits soit en Verilog soit en CHISEL.

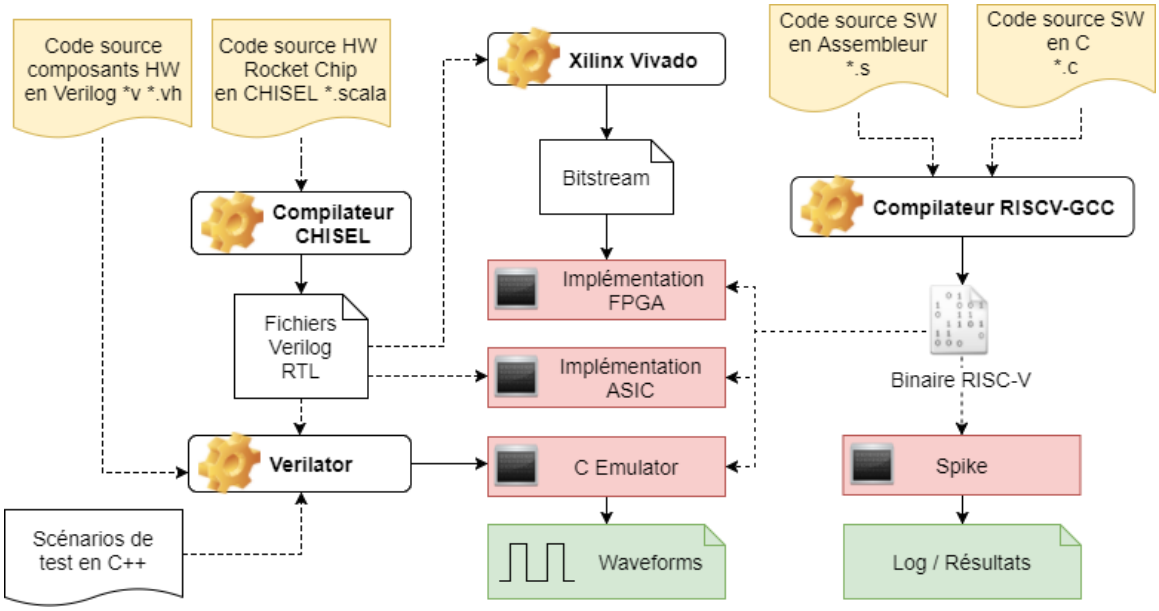

Fig.1. Flot de conception matériel (à gauche), et logiciel (à droite) sur la plateforme Rocket Chip. Les flèches en pointillées représentent les entrées, les flèches continues représentent les sorties. Les rectangles en rouge représentent les différentes cibles qui peuvent être visées par les exécutables binaires RISC-V.

Architecture du Rocket Chip

Les outils de la plateforme Rocket Chip permettent de générer (*cf.* figure 2) des architectures de systèmes sur puces composées principalement de :

1. tuiles (Rocket tiles) : regroupant un cœur Rocket avec ses mémoires cache de niveau L1 de données et d'instructions. Les tuiles peuvent toutes être du même type (structure homogène), et donc supportent les mêmes extensions d'instructions RISC-V standard (I/M/A/F/D/C) (11). Les tuiles peuvent aussi être différentes (structure hétérogène),
2. un bus système TileLink open source : il interconnecte les différents composants du système (périphériques standard ou personnalisés, contrôleurs d'interruptions, tuiles …),
3. une interface de débogage et programmation de type DTM (Debug Transport Module) ou JTAG.

Dans le cas où le SoC est implémenté sur FPGA/ASIC, un bus AXI4 permet la communication avec la mémoire principale (DDR3). Dans le cas d'une émulation, la mémoire principale est émulée par un modèle décrit en Verilog.

La plateforme matérielle est hautement paramétrable. Il est possible de configurer les paramètres internes : nombre de tuiles, nombre de cœurs, tailles et politiques des caches, paramètres de la MMU (PTW et TLB), activation ou désactivation de la FPU, ajout d'accélérateurs matériels dédiés (RoCC). Il est aussi possible de paramétrer les périphériques externes et les contrôleurs d'interruptions.

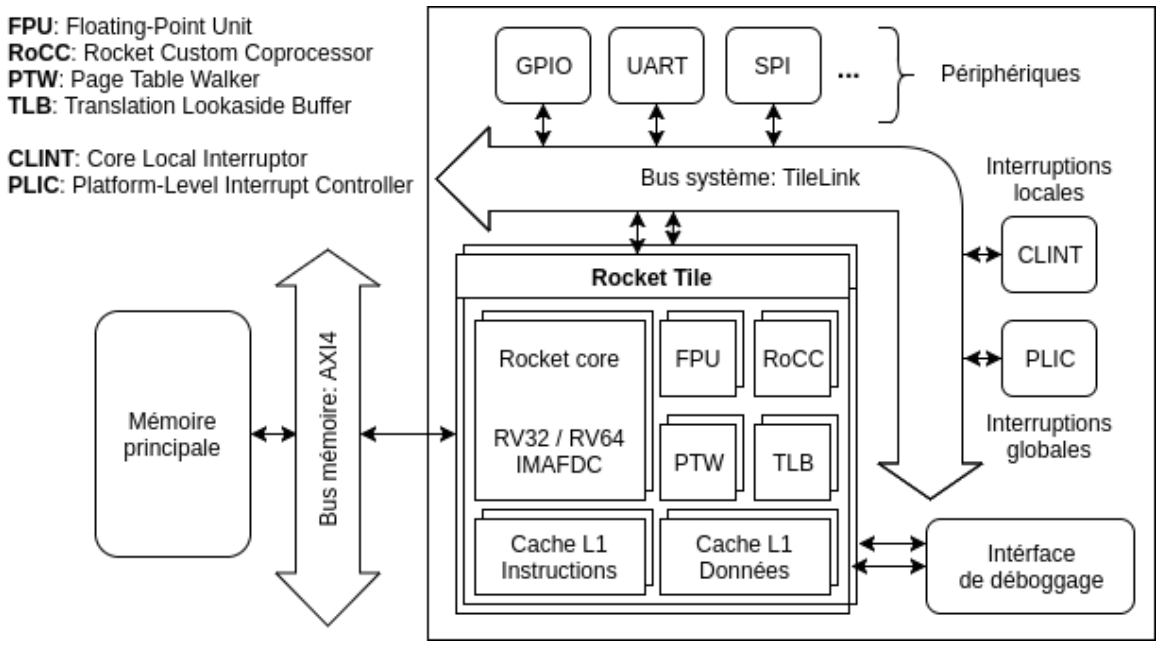

Fig.2. Architecture du système sur puce Rocket Chip généré.

III. Enseignement proposé

Le projet propose un enseignement d'une vingtaine d'heures encadrées sous forme de travaux pratiques, qui ont pour but d'illustrer et approfondir les concepts théoriques vus en cours.

Prise en main de l'environnement logiciel

Les étudiants se familiarisent avec l'environnement de développement, et mettent en œuvre les concepts de cross-compilation, assemblage, édition de liens, débogage et les techniques de script. Une synthèse des instructions en assembleur et des informations sur l'outil de simulation Spike leur est présentée. Des exemples de programmes sont étudiés et validés sur Spike. L'objectif ici est d'assimiler le flot de compilation, en incluant les fichiers d'initialisation (reset.S), l'éditeur de liens (link.ld) et les appels système (syscalls.c). Le flot enseigné est présenté dans la figure 3.

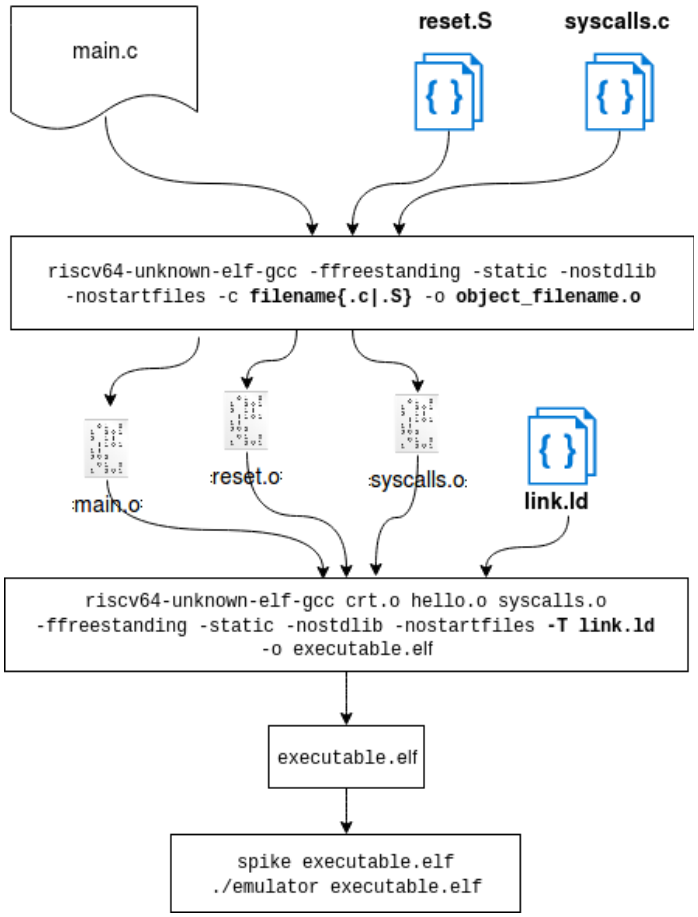

Fig.3. Flot de compilation en bare-metal.

Prise en main de l'environnement matériel

La seconde étape consiste à comprendre et utiliser le flot de conception matériel notamment en générant différents systèmes intégrés en faisant varier les paramètres architecturaux. Les étudiants découvrent l'intérêt d'utiliser un émulateur CABA, et de voir comment l'utiliser pour mesurer les temps d'exécution pour un programme assembleur donné. Ils étudient aussi comment observer et analyser les formes d'onde des signaux des composants internes du processeur pour chaque configuration matérielle.

Les étudiants appréhendent à travers cet exercice pratique la relation étroite entre matériel et logiciel. Comme exemple un morceau de programme manipulant des nombres en virgule flottante est étudié et testé sur deux types d'architectures matérielles : une qui contient une unité de calculs flottants (FPU) et une qui ne la contient pas. Ils arrivent donc à voir l'intérêt de modules matériels dédiés et les aspects à prendre en compte en compilant.

Les étudiants sont aussi amenés à écrire un programme qui écrit dans des adresses mémoires appartenant à l'espace d'adressage MMIO (memory-mapped input/output). Les formes d'ondes sont ensuite générées et analysés sur GTKWave.

Exceptions, interruptions et trappes

Cette partie plutôt de nature logicielle est consacrée à l'étude des événements qui peuvent interrompre l'exécution d'un programme : les exceptions, les appels systèmes et les interruptions avec mise en œuvre sur le processeur. Types d'interruptions, sources, accès aux registres de configuration, configuration des routines d'interruption (ISR) et

changement de contexte sont en particulier étudiés. Par souci de simplification, la configuration et la gestion des interruptions d'un timer sont prises comme exemple. Les notions apprises servent de base au TP suivant, puisque le gestionnaire d'interruptions sera adapté pour réaliser un ordonnanceur de taches basé sur les interruptions et le timer.

Multi-tasking

Les étudiants sont amenés dans cette partie à implémenter une application multitâche en assembleur sur un système monoprocesseur afin de maitriser les notions de temps partagé et de changement de contexte.

Un système simplifié de deux tâches périodiques concurrentes est implémenté. L'objectif de cette partie est d'assimiler les notions de sauvegarde/restauration de contexte et de gestion de la pile (stack) à travers l'utilisation de l'éditeur de lien. L'exemple est représenté dans la figure ci-dessous (Figure 4). Chaque tâche possède son propre contexte d'exécution :

- Mémoire de pile : à chaque tâche, un espace mémoire est alloué pour la pile. Leurs piles respectives sont les espaces d'adressages stack1 et stack2.
- Mémoire d'instructions : les deux tâches sont représentées par les deux fonctions task1() et task2().
- Sauvegarde des adresses des dernières instructions exécutées (PC, program counter), des dernières positions dans la pile (SP, stack pointer)), etc.

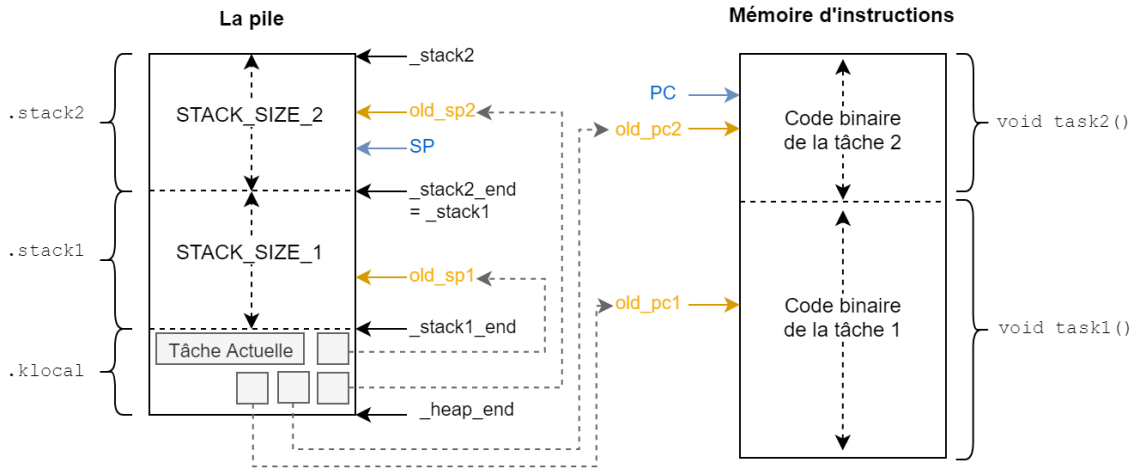

Fig.4. Exemple de systèmes à deux tâches.

Multi-processing cohérence mémoire et performance de caches

Dans une première partie, des architectures matérielles multiprocesseurs sont étudiées pour comprendre les mécanismes de cohérence mémoire et les défis qui en découlent lors de la conception de systèmes complexes.

Les étudiants sont amenés à manipuler deux types d'architectures, à savoir : une architecture à deux cœurs avec une mémoire cache cohérente et une autre double-cœur mais sans mécanismes de cohérence. Ensuite, en implémentant des scénarios d'accès en lecture/écriture pour chacun des deux processeurs, les étudiants mettent en évidence les problèmes de cohérence mémoire en testant les scénarios proposés pour les deux architectures. Dans une deuxième partie, les étudiants s'appuient sur l'outil Spike pour analyser les performances caches de binaires exécutables (*cf.* figure 5). Le but étant d'assimiler en pratique l'influence de l'emplacement des données dans la mémoire.

```
L2$ Bytes Read:            262464
L2$ Bytes Written:         0
L2$ Read Accesses:         8202
L2$ Write Accesses:        0
L2$ Read Misses:           8202
L2$ Write Misses:          0
L2$ Writebacks:            0
L2$ Miss Rate:             100.000%
D$ Bytes Read:             32768
D$ Bytes Written:          8
D$ Read Accesses:          8192
D$ Write Accesses:         1
D$ Read Misses:            8192
D$ Write Misses:           1
D$ Writebacks:             0
D$ Miss Rate:              100.000%
I$ Bytes Read:             147700
I$ Bytes Written:          0
I$ Read Accesses:          36925
I$ Write Accesses:         0
I$ Read Misses:            9
I$ Write Misses:           0
```

Fig.5. Exemple d'utilisation de Spike pour l'analyse des performances des caches L2, L1D et L1I

Développement et ajout de périphériques personnalisés

Afin de mieux appréhender le lien entre le matériel et le logiciel les étudiants sont amenés, à partir d'un ensemble de spécifications, à développer, intégrer, tester, simuler et débugger un périphérique matériel, en l'occurrence un modulateur en largeur d'impulsions (PWM). L'implémentation peut se faire en Verilog ou en CHISEL. Une couche d'abstraction du matériel (Hardware Abstraction Layer HAL) adéquate est développée et exploitée, sous forme de programmes C/assembleur, pour valider le fonctionnement du module ajouté. La figure 6 présente une capture d'écran représentant quelques signaux du périphérique matériel PWM ajouté.

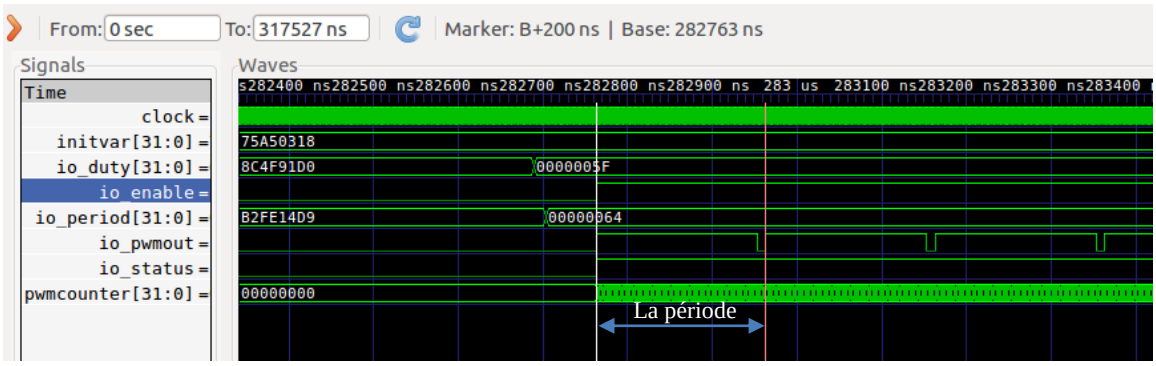

Fig.6. Signaux du module PWM ajouté.

Périphériques personnalisés et interruptions globales

Pour finir, les étudiants combineront leurs connaissances acquises au préalable pour implémenter dans le SoC un compteur en CHISEL (*cf.* figure 7), et l'interfacer avec le contrôleur d'interruptions globales (PLIC). Ensuite, une couche d'abstraction est développée et testée d'abord sans puis avec mise en œuvre du mécanisme d'interruptions. Un gestionnaire d'interruptions externes est développé en assembleur. Le test et le débogage est fait en analysant la sortie du compteur et les signaux du PLIC à l'aide de l'émulateur matériel intégrant ce nouveau compteur développé.

```
CounterInt( i ) := CounterOut && CounterIntEnable( i )
```

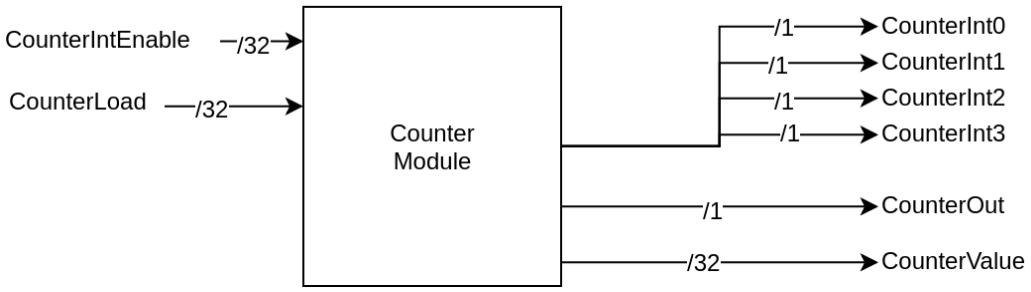

Fig.7. Signaux du module PWM ajouté.

Figure 8 et 9 illustrent respectivement les étapes de configuration des registres internes du compteur et le déclenchement de l'interruption. Le compteur décompte à partir de la valeur CounterLoad. Quand il arrive à zéro le signal de sortie CounterINT0, physiquement relié à l'entrée du PLIC, est mis à un, déclenchant alors une interruption globale.

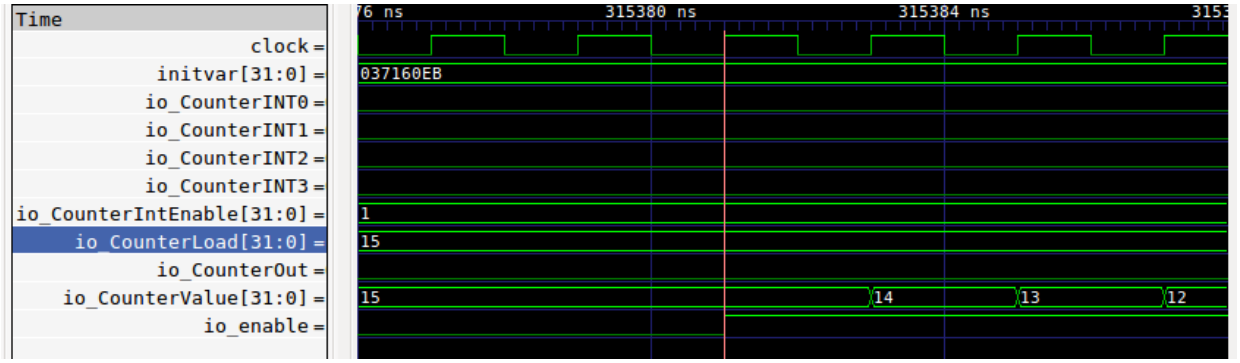

Fig.8. Configuration des registres du compteur.

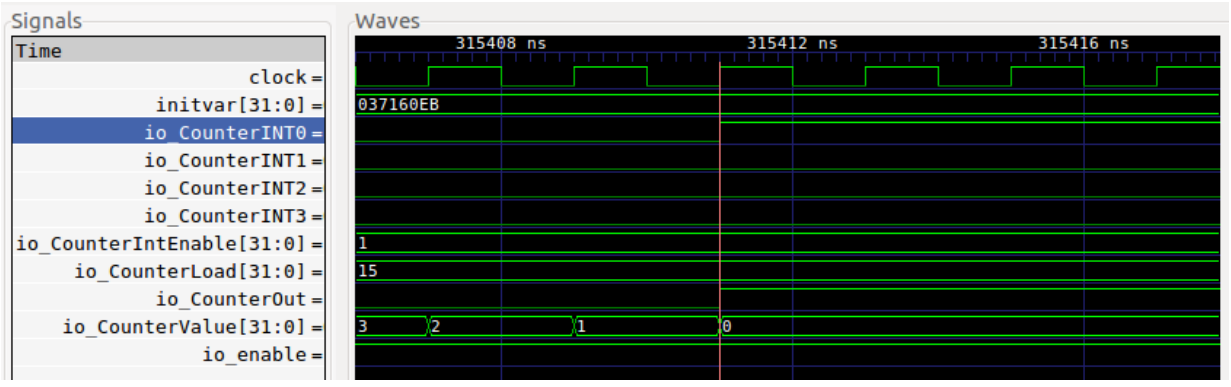

Fig.9. Interruption déclenchée au niveau du signal CounterINT0 reliée au PLIC.

Outils utilisés : Suite RISC-V GCC (GNU Compiler Collection), GDB (GNU Debugger) Make, Verilator, Spike, GTKWave

Environnement: Linux Ubuntu 16.04.

Langages de programmation logicielle (SW) : Langage C et assembleur RISC-V.

Langages de programmation/description matérielle (HW) : CHISEL, Verilog.

Conclusion

Cet article présente un nouvel enseignement en cours de mise en place à Grenoble INP dont l'objectif est de former des étudiants à la conception conjointe matérielle-logicielle de systèmes intégrés. L'objectif principal de ce nouvel enseignement est de donner à l'étudiant une vision plus claire de l'interaction entre le matériel et le logiciel et de l'impact d'un changement de l'un sur l'autre. De plus, la plateforme utilisée est libre de droit (Rocket Chip développée à Berkeley), elle est basée sur l'utilisation d'un processeur récent, le processeur RISC V et elle permet de faire du prototypage rapide et/ou d'aller jusqu'à l'implémentation physique sur FPGA ou ASIC. Cette flexibilité en fait un bon candidat comme outil pédagogique.

L'enseignement est actuellement proposé sous forme de travaux pratiques guidés. Il est à l'état expérimental et encore amené à évoluer pour intégrer d'autres notions, mais d'ores et déjà nous avons pour objectif de le transformer à moyen terme en projet moins guidé où les étudiants devront partir d'une application logicielle donnée et concevoir le meilleur système matériel qui supportera son exécution.

Remerciement

Ce travail a bénéficié d'une aide de l'Etat français au titre du Programme d'Investissements d'Avenir, IRT Nanoelec, portant la référence ANR-10-AIRT-05. Les outils utilisés sont mis à disposition dans le cadre du réseau GIP-CNFM et du projet IDEFI-FINMINA ANR-11-IDFI-0017.

Références

1. SoCLib Virtual Prototyping Platform, site web officiel: http://www.soclib.fr
2. L. Anghel, M. Benabdenbi et F. Petrot, "*System on chip project: Integration of a Motion-JPEG video decoder,*" 10th European Workshop on Microelectronics Education (EWME), Tallinn, 2014, pp. 167-170.
3. RISC-V Foundation, site web officiel: https://riscv.org/
4. Rocket Chip Generator, repository officiel : https://github.com/freechipsproject/rocket-chip
5. Wikipedia, https://fr.wikipedia.org/wiki/RISC-V
6. K. Asanović *et al.*, "*The Rocket Chip Generator,"* Technical Report No. UCB/EECS-2016-17, site web: https://people.eecs.berkeley.edu/~krste/papers/EECS-2016-17.pdf
7. C. Celio, D. Patterson, K. Asanović, *The Berkeley Out-of-Order Machine (BOOM) Design Specification*, University of California, Berkeley, December 4, 2016.
8. PULPino: An open-source microcontroller system based on RISC-V, site web officiel: https://pulp-platform.org
9. J. Bachrach *et al.,* CHISEL: Constructing Hardware in a Scala Embedded Language, University of California, Berkeley, Design Automation Conference 2012, site web: https://chisel.eecs.berkeley.edu/
10. A. Waterman, Y. Lee *et al. "Spike RISC-V ISA Simulator", repository officiel: https://github.com/riscv/riscv-isa-sim*
11. A. Waterman, K. Asanović, "*The RISC-V Instruction Set Manual, Volume II: Privileged Architecture,"* May 2017. Site web: https://riscv.org/specifications/privileged-isa/